**Python Generators**

In this tutorial, you'll learn how to create iterations easily using Python generators, how it is different from iterators and normal functions, and why you should use it.

**Generators in Python**

There is a lot of work in building an [iterator](https://www.programiz.com/python-programming/iterator) in Python. We have to implement a class with \_\_iter\_\_() and \_\_next\_\_() method, keep track of internal states, and raise StopIteration when there are no values to be returned.

This is both lengthy and counterintuitive. Generator comes to the rescue in such situations.

Python generators are a simple way of creating iterators. All the work we mentioned above are automatically handled by generators in Python.Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

**Create Generators in Python**

It is fairly simple to create a generator in Python. It is as easy as defining a normal function, but with a yield statement instead of a return statement.

If a function contains at least one yield statement (it may contain other yield or return statements), it becomes a generator function. Both yield and return will return some value from a function.

The difference is that while a return statement terminates a function entirely, yield statement pauses the function saving all its states and later continues from there on successive calls.

**def my\_gen():**

**n = 1**

**print('This is printed first')**

**# Generator function contains yield statements**

**yield n**

**n += 1**

**print('This is printed second')**

**yield n**

**n += 1**

**print('This is printed at last')**

**yield n**

**a = my\_gen()**

**"""**

**print(next(a))**

**print(next(a))**

**print(next(a))**

**print(next(a))**

**"""**

**for i in a:**

**print(i)**

**# generator:**

**# to generate prime numbers:**

**def is\_prime(m):**

**i = 2**

**while m % i != 0 :**

**i += 1**

**return i == m**

**def gen():**

**yield 2**

**yield 3**

**m = 5**

**while True:**

**if is\_prime(m) :**

**yield m**

**m += 2**

**g = gen()**

**# get next n primes**

**n = 25**

**for i in range(n):**

**print(next(g))**